Instructions

• This is an open-book/note exam. The exam length will be 24 hours

• There are three questions.

• Collaboration is not permitted in this exam.

• Use of the internet is not permitted in this exam.

• No question will be answered by the teaching staff unless it is typo-related.

• Sign the honor code statement on the front of this cover sheet.

• GOOD LUCK!

Honor Code

In recognition of and in the spirit of the Honor Code, I certify that I will neither give nor receive unpermitted aid on this examination and that I will report, to the best of my ability, all Honor Code violations observed by me.

Name: ______________________________________________________

Signature: __________________________________________________
1. Doctors Without Weekends

You’ve periodically helped the medical consulting firm Doctors Without Weekends on various hospital scheduling issues, and they’ve just come to you with a new problem. For each of the next \( n \) days, the hospital has determined the number of doctors they want on hand; thus, on day \( i \), they have a requirement that exactly \( p_i \) doctors be present.

There are \( k \) doctors, and each is asked to provide a list of days on which he or she is willing to work. Thus doctor \( j \) provides a set \( L_j \) of days on which he or she is willing to work.

The system produced by the consulting firm should take these lists and try to return to each doctor \( j \) a list \( L'_j \) with the following properties.

(A) \( L'_j \) is a subset of \( L_j \), so that doctor \( j \) only works on days he or she finds acceptable.

(B) If we consider the whole set of lists \( L'_1, \ldots, L'_k \) it causes exactly \( p_i \) doctors to be present on day \( i \), for \( i = 1, 2, \ldots, n \).

(a) Describe a polynomial-time algorithm that implements this system. Specifically, give a polynomial-time algorithm that takes the numbers \( p_1, p_2, \ldots, p_n \), and the lists \( L_1, \ldots, L_k \), and does one of the following two things.

- Return lists \( L'_1, L'_2, \ldots, L'_k \) satisfying properties (A) and (B); or
- Report (correctly) that there is no set of lists \( L'_1, L'_2, \ldots, L'_k \) that satisfies both properties (A) and (B).

(b) The hospital finds that the doctors tend to submit lists that are much too restrictive, and so it often happens that the system reports (correctly, but unfortunately) that no acceptable set of lists \( L'_1, L'_2, \ldots, L'_k \) exists.

Thus the hospital relaxes the requirements. They add a new parameter \( c > 0 \) and the system now should try to return to each doctor \( j \) a list \( L'_j \) with the following properties.

(A*) \( L'_j \) contains at most \( c \) days that do not appear on the list \( L_j \).

(B) (Same as before) If we consider the whole set of lists \( L'_1, \ldots, L'_k \) it causes exactly \( p_i \) doctors to be present on day \( i \), for \( i = 1, 2, \ldots, n \).

Describe a polynomial-time algorithm that implements this revised system. It should take numbers \( p_1, p_2, \ldots, p_n \), the lists \( L_1, L_2, \ldots, L_k \), and the parameter \( c > 0 \), and do one of the following things.

- Return lists \( L'_1, L'_2, \ldots, L'_k \) satisfying properties (A*) and (B); or
• Report (correctly) that there is no set of lists $L'_1, L'_2, \ldots, L'_k$ that satisfies both properties $(A^*)$ and $(B)$.

2. Truck Loading Problem

Suppose you’re acting as a consultant for the Port Authority of a small Pacific Rim nation. They’re currently doing a multi-billion-dollar business per year, and their revenue is constrained almost entirely by the rate at which they can unload ships that arrive in the port.

Here’s the basic sort of problem they face. A ship arrives, with $n$ containers of weight $w_1, w_2, \ldots, w_n$. Standing on the dock is a set of trucks, each of which can hold $K$ units of weight. (You can assume that $K$ and each $w_i$ is an integer) You can stack multiple containers in each truck subject to the weight restriction of $K$; the goal is to minimize the number of trucks that are needed in order to carry all the containers. This problem is NP-complete (you don’t have to prove this).

A greedy algorithm you might use for this is the following. Start with an empty truck, and begin piling containers 1, 2, 3, . . . into it until you get to a container that would overflow the weight limit. Now declare this truck “loaded” and send it off; then continue the process with a fresh truck. This algorithm, by considering trucks one at a time, may not achieve the most efficient way to pack the full set of containers into an available collection of truck.

(a) Give an example of a set of weights, and a value of $K$, where this algorithm does not use the minimum possible number of trucks.

(b) Show, however, that the number of trucks used by this algorithm is within a factor of 2 of the minimum possible number, for any set of weights and any value of $K$.

Hint: Consider the load of two trucks produced by the algorithm. Can they be combined into one?

3. Maximizing Ad Revenue Spread

You’re consulting for an e-commerce site that receives a large number of visitors each day. For each visitor $i$, where $i \in \{1, 2, \ldots, n\}$, the site has assigned a value $v_i$, representing the expected revenue that can be obtained from this customer.

Each visitor $i$ is shown one of $m$ possible ads $A_1, A_2, \ldots, A_m$ as he or she enters the site. The site wants a selection of one ad for each customer so that each ad is seen, overall, by a set of customers of reasonably large total weight. Thus, given a selection of one ad for each customer, we will define the spread of this selection to be the minimum, over $j = 1, 2, \ldots, m$, of the total weight of all customers who were shown ad $A_j$. 
**Example:** Suppose there are six customers with values 3, 4, 12, 2, 4, 6, and there are $m = 3$ ads. Then, in this instance, one could achieve a spread of 9 by showing ad $A_1$ to customers 1, 2, 4, ad $A_2$ to customer 3, and ad $A_3$ to customers 5 and 6.

The ultimate goal is to find a selection of an ad for each customer that maximizes the spread. Unfortunately, this optimization problem is NP-hard (you don’t have to prove this). So instead, we will try to approximate it.

(a) Give a polynomial-time algorithm that approximates the maximum spread to within a factor 2. That is, if the maximum spread is $s$, then your algorithm should produce a selection of one ad for each customer that has spread at least $s/2$. In designing your algorithm, you may assume that no single customer has a value that is significantly above the average; specifically, if $\bar{v} = \sum_{i=1}^{n} v_i$ denotes the total value of all customers, then you may assume that no single customer has a value exceeding $\bar{v}/(2m)$.

(b) Give an example of an instance on which the algorithm you designed in part (a) does not find an optimal solution (that is, one of maximum spread). Say what the optimal solution is in your sample instance, and what your algorithm finds.